Markus Huhtamäki

MOBIILISOVELLUS SEKÄ SERVERIPÄÄN OHJELMISTO KOKOUSTIETOJEN LUONTIIN

Tekniikka
2018

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Markus Huhtamäki
Opinnäytetyön nimi	Mobiilisovellus sekä serveripään ohjelmisto kokoustietojen luontiin.
Vuosi	2018
Kieli	suomi
Sivumäärä	35
Ohjaaja	Pirjo Prosi

Tämä työ on Vaasan ammattikorkeakoulun tietotekniikan koulutusohjelman päättötyö. Opinnäytetyö on suunniteltu yhdessä Vaasan ammattikorkeakoulun yliopettaja, FT Ghodrat Moghadampourin kanssa. Työn tavoitteena oli luoda mobiilisovellus sekä serveripään ohjelmisto kokoustietojen tallentamiseen, selaamiseen sekä muokkaamiseen.

Työ aloitettiin kartoittamalla ohjelmistojen tekemiseen tarvittavat työkalut sekä sopivimmat teknologiat. Palvelinpään ohjelmisto toteutettiin käyttämällä Spring Boot nimistä ohjelmistokehystä. Mobiilisovelluksessa käytettiin hyväksi kirjastoa nimeltä Retrofit pyyntöjen lähettämiseksi palvelimelle.

Varsinaisen työn toteutus aloitettiin suunnittelemalla serveripään ohjelmiston arkkitehtuuri sekä tietokannan rakenne. Mobiilisovelluksen käyttöliittymä suunniteltiin myös valmiiksi ennen varsinaiseen toteutukseen siirtymistä.

Työn tuloksena syntyi toimiva ja nykyaikainen client-server ohjelmistoratkaisu, jolla työn vaatimukset saatiin toteutettua.

Avainsanat mobiili, restful , spring, java

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information technology

ABSTRACT

Author	Markus Huhtamäki
Title	Mobile Application and Server Software for creating and persisting Meeting Information.
Year	2018
Language	Finnish
Pages	35
Name of Supervisor	Pirjo Prosi

The purpose of this thesis was to create a mobile application as well as a server backend software for persisting, fetching and editing meeting information. The idea for the thesis came from Mr Ghodrat Moghadampour, Ph.D of Vaasa University of Applied Sciences.

The work was started by studying different technologies and tools to find most useful and up-to-date methods for this project. The server software was implemented using a framework called Spring Boot. For the mobile application, a library called Retrofit was used to create the requests to the server.
The actual implementation was started by designing the server architecture as well as database structure. The mobile application was also drafted before the actual implementation.

As a result of this thesis, a working and modern client-server solution was created.

Keywords mobile, restful , spring, java

SISÄLLYS

LYHENTEET JA TERMIT

Push-notifikaatio	Android-sovelluksen ilmoitus
Basic-autentikaatio	Käyttäjän tunnistautumistapa
REST	Representational State Transfer, HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.
API	Application programming interface, Ohjelmointi rajapinta
HTTP	HyperText Transfer Protocol, Hypertekstin siirto protokolla
XML	Extensible Markup Language, Laajennettava merkintäkieli
JSON	JavaScript Object Notation.
Framework	Ohjelmistokehys
Autentikaatio	Käyttäjän tunnistautuminen palveluun
Autorisaatio	Palveluiden rajaaminen
JDBC	Java Database Connectivity, Javan tietokanta rajapinta.
IoC	Inversion of Control, Hallinnan inversio
MVC	Model-View-Controller, Malli-Näkymä-Ohjain
Design pattern	Suunnittelumalli
URL	Uniform Resource Locator

REST Controller	REST-ohjain
Metodi	Java-luokassa ajettava tehtävä
Business layer	Bisnestaso
Persistence layer	Tallennustaso
JPA	Java Persistence API
Repository	Säilö
SQL	Structured Query Language, Relaatiokanto-jen kyselykieli
Container	Säiliö
Fragment	Päänäkymän päälle aukeava pienoisnäkymä

1 JOHDANTO

1.1 Opinnätetyön taustaa

Tämä opinnäytetyö on suunniteltu yhdessä Vaasan ammattikorkeakoulun yliopettajan, FT Ghodrat Moghadampourin kanssa. Ennen tämän työn alulle laittamista, ilmaisin hänelle haluni tehdä opinnäytetyön minua kiinnostavasta projektista, jota olin jo hieman aloittanut. Tämä projekti oli mobiilisovellus, jonka avulla voitiin siirtää haluttujen mobiilisovelluksien push-notifikaatioita serverille. Tätä ideaa hieman soveltamalla luonnosteltiin tämän opinnäytetyön aihe sekä sille toiminnalliset vaatimukset.

1.2 Mobiilisovellus

Nykyajan kiihtyvässä maailmassa, jossa ihmisten kiire vain lisääntyy, on mobiilisovellusten suosio kasvanut rajusti. Sovellukset on oltava nopeasti saatavilla sekä helposti käytettäviä. Siksi onkin luontevaa, että ihmiset suosivat mobiilisovelluksia, jotka ovat aina mukana puhelimen muistiavaruudessa. Siitä syystä päädyinkin tekemään mobiilisovelluksen kokoustietojen tallentamiseen PC-sovelluksen sijaan.

Työni tavoitteeksi asetettiin mobiilisovelluksen sekä serveriohjelmiston kehittäminen kokoustietojen käsittelyyn. Mobiilisovelluksessa täytyi olla ominaisuus jonka avulla voi luoda kokouksia, joihin on mahdollista lisätä puhelimeen tallennettujen yhteystietojen avulla osallistujia. Näitä kokouksia piti pystyä myös hakemaan serveriltä eri kriteerejen avulla, esimerkiksi osallistujan nimen perusteella. Koska yhdelllä käyttäjällä ei todennäköisesti tule olemaan kerrallaan enempää kuin ehkä kymmenkunta kokousta, jätettiin tämä eri kriteereillä onnistuva haku pois ja sovelluksessa haetaan aina kaikki käyttäjälle tallennetut kokoukset.

Kokouksille tallennettujen perustietojen, kuten otsikon, paikan, päivämäärän ja kellonajan lisäksi, oli luotava myös mahdollisuus tallentaa kokouskohtaisia tiedostoja, joita mahdollisesti luodaan kokouksen yhteydessä. Näitä tiedostoja

voivat olla mm. puhelimen kameralla otetut valokuvat, kirjoitetut tekstitiedostot tai muut muistiinpanot. Näitä serverille tallennettuja tiedostoja oli myös pystyttävä lataamaan puhelimeen tarvittaessa.

Jotta tärkeät kokoukset eivät menisi vahingossa ohi, tuli kokouksille olla mahdollista myös ajastaa hälytys, joka muistuttaa käyttäjää tulevista kokouksista. Tämä hälytys huomauttaa käyttäjää merkkiäänellä sekä ponnahdusikkunalla, josta näkyy ko. kokouksen tiedot.

Tietoturva otettiin myös sovelluksessa huomioon. Jotta kokoustiedot eivät joutuisi asiattomien käsiin, sovelluksessa toteutettiin käyttäjän rekisteröinti sekä sisäänkirjautuminen. Käyttäjäkohtainen salasana tallennettiin tietokantaan kryptatussa muodossa, mikä lisää tietoturvaa. Mobiilisovelluksen ja serverin välisessä liikenteessä käytettiin Basic-autentikaatiotunnistautumista mikä varmistaa, että ilman rekisteröityä käyttäjätunnusta ja salasanaa ei pääse mihinkään tietokannassa oleviin tietoihin käsiksi.

1.3 Palvelinohjelmisto

Jotta mobiilisovelluksella luotuja kokouksia voitaisiin tallentaa tietokantaan, on oltava myös palvelinohjelmisto, joka hoitaa varsinaisen tiedon käsittelyn. Tässä työssä loin palvelinohjelmiston käyttämällä REST -arkkitehtuuria. Yleisesti tällaisesta web-palvelusta puhuttaessa käytetään nimitystä RESTful API.

REST on lyhennys englanninkielisistä sanoista Representational State Transfer. Se on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen /1/. Seuraavassa luvussa kerron tarkemmin REST -rajapinnasta.

2 RESTFUL API – PALVELINOHJELMISTO

Tässä osiossa käsitellään REST -arkkitehtuuria ja sen historiaa lähteen 2 pohjalta.

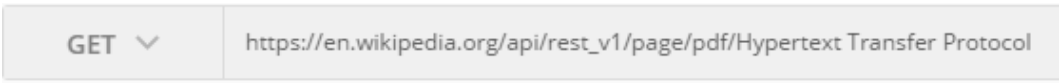
2.1 REST

REST-arkkitehtuurin loi vuosina 1996-1999 mies nimeltä Roy Fielding. Lyhenne REST alkoi vakiintua arkkitehtuurin lyhenteenä vasta vuoden 2000 jälkeen, jolloin herra Fielding julkaisi tieteellisen tutkielmansa ”Architectual styles and the design of network-based software architectures”.

RESTful API web-palvelussa tiedonsiirto on pohjautuu HTTP -protokollaan. Tästä johtuen tiedonsiirrossa käytetään ns. HTTP -verbejä, joilla kyselyn muodostava laite ilmaisee minkälainen pyyntö on kyseessä. Yleisimmät verbit ovat:

- GET, käytetään resurssien hakua varten
- POST, minkä tahansa resurssin lähettämiseen palvelimelle
- PUT, käytetään kun tallennetaan/päivitetään vain yksi resurssi
- DELETE, tietyn resurssin poistamiseen

Näitä verbejä käyttämällä kyselyn suorittava laite tekee ns. requestin eli pyynnön palvelimelle jonkin operaation suorittamista varten. Tämä pyyntö tehdään johonkin tiettyyn URL -osoitteeseen minkä palvelin tarjoaa. Kuvassa 1 esimerkki tällaisesta pyynnöstä.



Kuva 1. GET-pyyntö

Kuvassa 1 suoritetaan GET-pyyntö ko. osoitteeseen, jolloin saadaan vastauksena dokumentaatio pdf-muodossa HTTP-protokollasta. Tässä tapauksessa palvelimelta tuleva data on binäärimuotoista.

REST -palvelimet tukevat usein erilaisia tiedostomuotoja joissa vastauksen voi palvelimelta saada. Yleisimmä muodot ovat tekstimuoto, XML sekä JSON.

REST-arkkitehtuurin suurimpiin etuihin kuuluu tilattomuus. Tilattomuudella tarkoitetaan clientin eli palvelimeen yhteyttä ottavan laitteen sekä palvelimen välistä tilattomuutta. Yhteyttä ottavan laitteen eikä palvelimen tarvitse olla tietoinen mahdollisista edellisistä yhteydenotoista, vaan palvelin pystyy tarjoamaan palveluitaan riippumatta siitä, mitä on tapahtunut aiemmin. Tämä mahdollistaa todella hyvän skaalautuvuuden kasvavalle määrälle laitteita. Nykyaikaisissa pilvipalveluissa pyörivät palvelinohjelmistot voidaan näin skaalata vastaamaan lähes rajatonta määrää kyselyitä tekeviä laitteita.

REST-arkkitehtuuriin rakennettu palvelinohjelmisto käyttää myös yhdenmukaista käyttöliittymää. Tämän ansiosta palvelinohjelmiston peruskäyttöön liittyvää toimintaa ei tarvitse erikseen dokumentoida vaan mikä tahansa laite, joka pystyy luomaan HTTP -kyselyitä, pystyy kommunikoimaan palvelimen kanssa. Ainoa asia mitä kyselyitä suorittavan laitteen on osattava, on käsitellä palvelimelta tulevaa dataa.

2.2 Spring Framework

Tämän työn palvelinohjemisto ohjelmoitiin Java-ohjelmointikielellä. Ensimmäinen versio Java-ohjelmointikielestä julkaistiin vuonna 1995, jonka jälkeen Java on kasvanut yhdeksi suosituimmista ohjelmointikielstä. Javan suosion ansiosta Javalle löytyy paljon erilaisia ohjelmistokehyksiä, jotka helpottavat ja nopeuttavat ohjelmistojen kehitystä sekä tarjoavat erilaisia työkaluja web-ohjelmiston kehitykseen. Näihin ohjelmistokehyksiin viitatessa käytetään englannin kielestä vakiintunutta termiä framework. Jatkossa tullaan käyttämään termiä framework viitatessa näihin ohjelmistokehyksiin.

Yksi suurimmista ja suosituimmista frameworkeistä on amerikkalaisen Pivotal Softwaren kehittämä Spring Framework. Spring sisältää lukuisia moduuleja, jotka tarjoavat erilaisia palveluita tiettyihin tehtäviin. Alla kerrotaan tarkemmin muutamasta tärkeimmästä moduulista.

2.2.1 Spring Security

Spring Security tarjoaa työkalut web-applikaation tietosuojaan. Tietosuoja muodostuu kahdesta osiosta:

- Autentikaatio – Käyttäjän tunnistautuminen järjestelmään. Käyttäjätunnus sekä salasana validoidaan tietokannassa olevia tunnuksia vastaan./3/
- Autorisaatio – Käyttäjälle määritetään pääsy sille sallittuihin järjestelmiin./4/

2.2.2 Data access

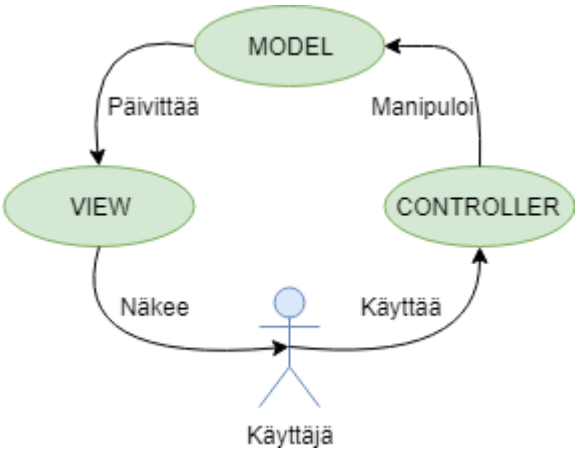
Data access mahdollistaa tietokantojen helpon integroimisen web-palveluun. Erilaisiin tietokantoihin voidaan muodostaa yhteys käyttämällä data access-moduulin tarjoamaa JDBC API -tekniikkaa./5/

2.2.3 Inversion of Control, IoC

Inversion of control on yksi keskeisimpiä Spring frameworkin moduuleista. Ns. dependecy injection –tekniikan avulla web-applikaation kehittäjä voi jättää applikaatiossa olevien palveluiden luonnin Spring frameworkin huoleksi ja siten käyttää arvokkaan aikansa varsinaisen applikaation kehittämiseen. Osiossa ”Toteutus” annetaan tarkempi esimerkki tästä tekniikasta./6/

2.2.4 Model-View-Controller, MVC

Tämän moduulin avulla voidaan käyttää ns. model-view-controller design-patternia. Tämä suunnittelumalli on yleinen web-applikaatioiden arkkitehtuurimalli. Kuvassa 2 esitetään MVC-pattern./7/



Kuva 2. MVC pattern

2.2.5 Testing

Koska jokainen itseään kunnioittava modernin web-applikaation kehittäjä käyttää työssään ns. Test-Driven-Development-teknikkaa, jossa ohjelmoitavalle logiikalle kehitetään ensin testauskoodi, joka sitten testaa koodatun logiikan, tarjoaa Spring Framework moduulin yksikkö- sekä integraatiotestien kirjoittamiseen./8/

2.3 Spring Boot

Tämän työn palvelinohjelmiston tuottamiseen käytettiin Spring Frameworkin versiota Spring Boot. Tämä versio Spring frameworkistä on kehitetty mahdollisimman tuotantovalmiiksi kokonaisuudeksi, jotta ohjelmistokehittäjän tarvitsisi tehdä mahdollisimman vähän erilaisia konfiguraatioita ja alustavia toimenpiteitä, että ohjelmiston saa pisteeseen, jonka jälkeen voi keskittyä itse ominaisuuksien kehittämiseen./9/

Spring Bootin mukana tulee sisäänrakennettuna Tomcat-palvelinohjelmisto. Tämä jälleen nopeuttaa ohjelmistokehitystä, koska ei tarvitse itse asentaa ja konfiguroida palvelinta ohjelmiston pyörittämiseen. Tätä palvelinta pystyy myös helposti vaihtamaan määrittelemällä vain yhdessä konfiguraatiotiedostossa palvelimen nimen./10/

3 MOBIILIAPPLIKAATIO

3.1 Yleistä mobiilikehityksestä

Nykyaikainen mobiiliapplikaatiokehitys nojautuu pitkälti JavaScript-pohjaisiin ratkaisuihin. Tämä on tulosta JavaScript -ohjelmointikielen kasvaneesta suosiosta. JavaScript-kielen käyttö mobiilikehityksessä mahdollistaa applikaation kehityksen useille laitealustoille samanaikaisesti. Tämä nopeuttaa ohjelmistokehitystä ratkaisevasti sekä mahdollistaa mobiilikehittäjien keskittymisen vain yhteen ohjelmointikieleen. Tämän ansiosta applikaation kehitys onnistuu pienemmällä määrällä kehittäjiä sekä puolittaa ohjelmiston ylläpitoon menevän ajan.

Kuitenkin, jos on tarkoituksena kehitää applikaatio vain tietylle alustalle, esimerkiksi Androidille, on järkevämpää ohjelmoida laitealustan natiivilla ohjelmointikielellä sekä käyttää sille löytyviä natiiveja kirjastoja, joita Androidin kehittäjä Google tarjoaa.

Koska tämän työn aikataulu oli hieman tiukka ja minulla oli aikaisempaa kokemusta natiivista Android-ohjelmoinnista, päädyin kehittämään mobiiliapplikaation natiivisti Androidille.

3.2 Retrofit 2 –kirjasto

Aloittaessani kehittämään mobiiliapplikaatiota, ryhdyin ensin tutkimaan tekniikoita ja työkaluja, joita tällaisen applikaation kehittäminen edellyttää. Nopeasti törmäsin kirjastoon nimeltä Retrofit 2, joka on kehitetty juuri Android-applikaation ja RESTful API:n väliseen kommunikointiin. Tälle kirjastolle löytyi myös kattava dokumentaatio sekä selkeitä oppaita, minkä avulla kehittää ohjelmistoa, joten oli helppo valinta ottaa tämä kirjasto mukaan ohjelmistokehitykseen.

3.2.1 Retrofit 2 Java interface

Retrofit-kirjaston avulla perinteinen HTTP API voidaan korvata normaalilla Java interfacellä. Kuvassa 3 on esimerkki interfacestä, jota käytin työssäni uuden käyttäjän rekisteröintiin./10/

```
public interface MeetingAppRetrofitClient {

    @POST("sign-up/")
    Call<ResponseBody> register(@Body ApplicationUser user);
```

Kuva 3. Retrofit 2 interface

Kuvasta 3 näemme seuraavaa:

- Kyselyn tyyppi – POST
- URL osoite, johon kysely osoitetaan – ”sign-up/”
- Kyselyn bodyyn lisätty käyttäjäobjekti ApplicationUser user
- Kyselyn palautustyyppi – ResponseBody

Ylläolevassa tapauksessa interface oli hyvin yksinkertainen. Monimutkaisempiin tarkoituksiin Retrofit tarjoaa monipuoliset työkalut, joista alla muutamia:

- URL-manipulaatio
  Kyselyn URL –osoitetta voidaan muokata tarvittaessa dynaamisesti käyttämällä parametrejä. Esimerkiksi: ”users/getUserById/{id}”, palauttaa käyttäjän annetulla id-numerolla.

- Query-parametrien käyttö
  Kyselyyn voidaan lisätä ns. Query parametrejä, joilla voidaan esimerkiksi järjestetää haetut objektit haluttuun järjestykseen.

- Header-manipulaatio
  Tapauksessa, jossa käyttäjä tunnistautuu palveluun käyttäjätunnuksen ja salasanan avulla, voidaan kyselyn Header –osioon lisätä tarvittavat tunnukset tunnistautumista varten.

- Multipart data
  Interfacessä luodut kyselymetodit voidaan merkitä lähettämään multipart dataa. Tätä käytetään tiedostoja lähetettäessä palvelimelle./10/

3.2.2 Retrofit 2 HTTP kyselyn luonti

Kun edellissä osiossa kuvattu Retrofit–interface on luotu, voidaan siirtyä itse HTTP –kyselyn luontiin./10/

```
Retrofit builder = new Retrofit.Builder()
        .baseUrl("localhost:8080/rest/v1/")
        .build();
MeetingAppRetrofitClient client = builder.create(MeetingAppRetrofitClient.class);
```

Kuva 4. Retrofit builder

Retrofit APIn mukaisesti luodaan ensin Retrofit.Builder-objekti, jolla sitten luodaan implementaatio edellä tehdystä MeetingAppRetrofitClient –luokasta. Kun implementaatio on luotu, voidaan käynnistää varsinainen HTTP-kysely.

```
Call<ResponseBody> call = client.register(user);
call.enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {

    }

    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {

    }
});
```

Kuva 5. Retrofit HTTP-kysely

Retrofit –kyselyn ajossa implementoidaan interface Callback. Tämä lisää kaksi callback –metodia, jotka suoritetaan riippuen kyselyn tuloksesta. Metodi onResponse ajetaan, jos kysely sai vastauksen palvelimelta, oli se sitten mikä vastaus tahansa. Jos esimerkiksi internet yhteys on poikki eikä palvelimeen saada

yhteyttä, ajetaan metodi onFailure. Riippuen käyttäjän tarpeista, näissä kahdessa metodissa voidaan suorittaa erilaisia tehtäviä, mitä ikinä tarvitaankin.

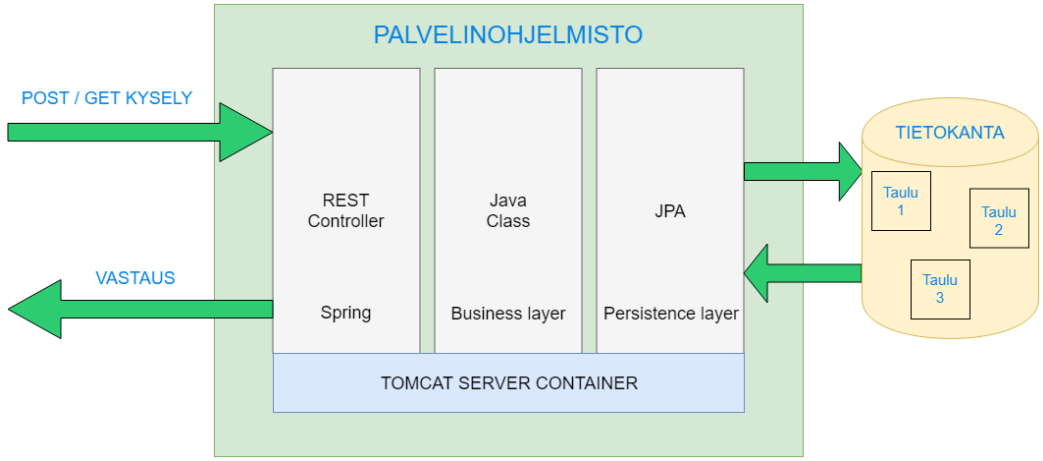
Retrofit-kirjaston etuihin kuuluu automaattinen asynkroninen HTTP-kyselyjen ajo. Tämä tarkoittaa, että jokainen kysely, joka ajetaan metodilla .enqueue, ajetaan automaattisesti uudessa säikeessä/10/. Lyhyesti selitettynä; Tietotekniikassa sanalla säie tarkoitetaan pientä, kevyttä prosessia. Tällaisia pieniä prosesseja voidaan ajaa rinnakkain tehden prosessoinnista tehokkaampaa sekä nopeampaa. Minkä tahansa käyttöliittymän ohjelmoinnissa tämä on tärkeää, ettei säie, jossa käyttöliittymää ajetaan, kuormitu liikaa. Jos kuormitusta on liikaa, käyttöliittymä voi jumittua kokonaan, jolloin käyttäjän on odotettava, että käynnistetty prosessi on valmis ennen kuin hän voi jatkaa ohjelmiston käyttöä.

4 TOTEUTUS

4.1 Palvelinohjelmisto

4.1.1 Arkkitehtuurin suunnittelu

Kuvassa 5 kuvataan palvelinohjelmiston arkkitehtuuria.



Kuva 5. Palvelinohjelmistoarkkitehtuuri

Palvelinohjelmiston teko alkoi arkkitehtuurisuunnitelmalla, josta muodostui kuvan 5 mukainen kokonaisuus. Tämän hahmotelman mukaan lähdettiin toteuttamaan palvelinohjelmistoa. Tiedonkulku tapahtuu alla olevan järjestyksen mukaisesti.

1. REST Controller

   Käyttäjältä tulevat kyselyt otetaan vastaan REST Controller -luokissa, joista data siirtyy seurvaavalle, business layer-tasolle. Kyselyt osoitetaan controllerin eri metodeille, riippuen kyselyn URL-osoitteesta.

2. Business layer

   Termillä business layer tarkoitetaan ohjelmiston tasoa, jossa määritellään mitä tulleelle datalle tehdään, kuinka se tallennetaan ja kuinka se on yhteydessä muihin ohjelmiston osiin. Business layerin sisällä olevaa logiikkaa kutsutaan liiketoimintalogiikaksi. Bisneslogiikassa määritellään

erilaisia palveluita ja ohjelmiston eri komponentteja. Näitä palveluita käytetään sitten datan käsittelyyn.

3. Persistence layer, JPA

   Tämä ohjelmiston taso on vastuussa itse datan tallennuksesta, hakemisesta sekä päivittämisestä tietokantaan. Koska Spring boot framework on valmiiksi konfiguroitu mahdollisimman pitkälle, tulee sen mukana vakiona spring-data-jpa-moduuli. Tämä moduuli mahdollistaa JPA repository interfacen käytön. Nämä repository interfacet tekevät datan tallennuksen ja hakemisen eri kriteereillä tietokannasta helpoksi. Yhtään riviä SQL-koodia ei tarvitse kirjoittaa itse vaan Spring hoitaa kaiken puolestasi.

4. Tietokanta

   Data joko tallennetaan tai sitä haetaan tietokannasta. Tämä data on tallennettuna tietokannassa oleviin tietokantatauluihin. Näillä tietokantatauluilla voi olla erilaisia suhteita ja ne voivat olla kytkettynä toisiinsa erilaisilla avaimilla. Tietokannasta haettu data siirretään eteenpäin käyttämällä edellisessä osassa mainittuja repository interface-objekteja josta se palaa samaa reittiä takaisin käyttäjälle vastauksena kyselyyn.

Koko palvelinohjelmisto pyörii serverin eli palvelincontainerin sisällä. Tässä työssä käytettiin Tomcat-palvelinta. Tomcat tulee oletuksena Spring boot-applikaation mukana, mutta sen voi vaihtaa myös johonkin toiseen containeriin jos haluaa.

4.1.2 Palvelinohjelmiston toteutus

Palvelinohjelmistoa lähdettiin toteuttamaan tiedonkulkujärjestyksessä. Ensimmäisenä toteutettiin controllerit, joilla tulevat kyselyt otetaan vastaan. Kuvassa 6 kuvataan käyttäjän rekisteröintiä.

```
@Autowired
private CustomUserDetailsService service;

@PostMapping("/{user}")
private ResponseEntity<?> registerUser(@RequestBody ApplicationUser user){
    if(userRepo.findByUsername(user.getUsername()) == null) {
        service.save(user);
        return new ResponseEntity<>(HttpStatus.OK);
    }
    return new ResponseEntity<>(HttpStatus.CONFLICT);
}
```

Kuva 6. Käyttäjän rekisteröinti

Uusi käyttäjä tulee palvelimelle kyselyn body -osiossa. Seuraavaksi tarkistetaan löytyykö tietokannasta jo samalla nimellä rekisteröity käyttäjä. Tämä tarkastus tehdään hakemalla kannasta käyttäjää uuden käyttäjän nimellä. Jos käyttäjää ei löydy, tallennetaan se kantaan CustomUserDetailsServicen avulla. Tämä palvelu salaa käyttäjän salasanan ja tallentaa käyttäjän kantaan. Jos kannasta löytyy jo samalla nimellä rekisteröity käyttäjä, annetaan kyselyn tekevälle laitteelle virheilmoitus.

Kaiken datan tallennus sekä haku tapahtuu käyttäen JpaRepository rajapintoja. Kuvassa 7 kuvataan JpaRepository rajapintaa, joka huolehtii käyttäjien tallennuksesta sekä hausta. Tallennus- tai poistometodeita ei tarvitse erikseen määritellä, ne ovat oletuksena osa JpaRepository -luokan objektia.

```
public interface ApplicationUserRepository extends JpaRepository<ApplicationUser, Long> {

    ApplicationUser findByUsername(String username);
}
```

Kuva 7. ApplicationUserRepository

4.1.3 Tietokanta

Tässä työssä käytettiin tietokantana MySQL -tietokantaa. MySQL on ns. relaatiotietokantaohjelmisto. Relaatiotietokannat perustuvat tietokannassa olevien taulujen keskinäisiin suhteisiin. Näitä suhteita on kolmea tyyppiä.

- Yksi-yhteen

  Esimerkkinä kuvittellinen tilanne, jossa on tietokantataulut ”Asiakas” sekä ”Osoite”. Yhdellä asiakkaalla on vain yksi osoite. Näin nämä muodostavat suhteen yksi-yhteen.

- Yksi-moneen

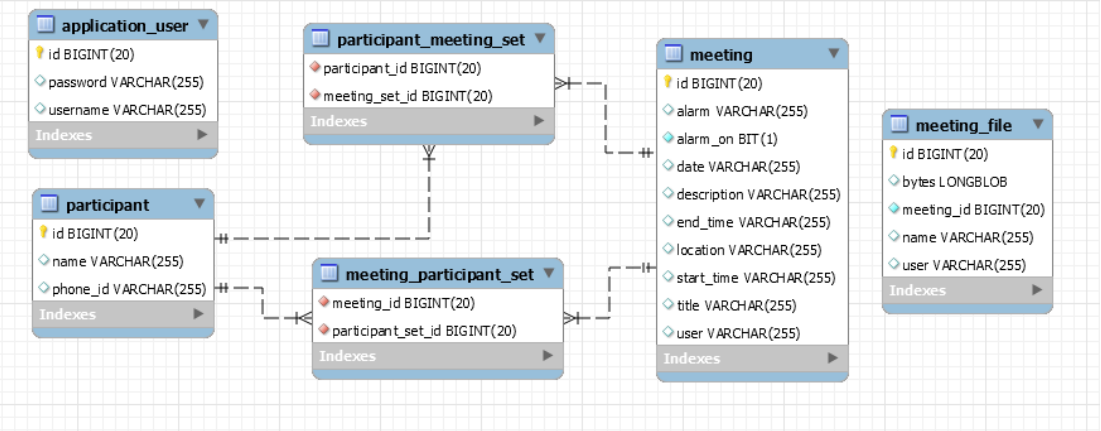
  Esimerkkinä tilanne, jossa ovat tietokantataulut, ”Asiakas” ja ”Tilaus”. Asiakas voi suorittaa monia tilauksia, joten asiakastaulussa voi olla tiedot monista tilauksista. ”Tilaus”-taulussa puolestaan on vain tieto mille asiakkaalle se kuuluu. Näin muodostuu yksi-moneen suhde.

- Moni-Moneen

  Esimerkkinä tilanne, jossa ovat tietokantataulut ”Tilaus” ja ”Tuote”. Kummallakin, tilauksella sekä tuotteella voi olla useampi kappale toisiaan. Tilaukseen voi kuulua useita tuotteita, sekä samaa tuote voi kuulua useampaan tilaukseen. Näin muodostuu moni-moneen suhde.

Nämä eri suhteissa olevat tietokantataulut liitetään toisiinsa käyttäen ns. tietokanta-avaimia. Näitä avaimia on kahden tyyppisiä, Primary key (PK) sekä Foreign key (FK). Kaikissa tietokannan tauluissa määritellään primary key-avain, jonka perusteella se voidaan liittää johonkin toiseen tietokannan tauluun. Näin taulut muodostavat PK – FK liitoksia.

Seuraavaksi kuvaataan työssä tehdyn tietokannan rakennetta tarkemmin. Kuva 8 kuvaa tietokannan rakennetta.



Kuva 8. Tietokantarakenne

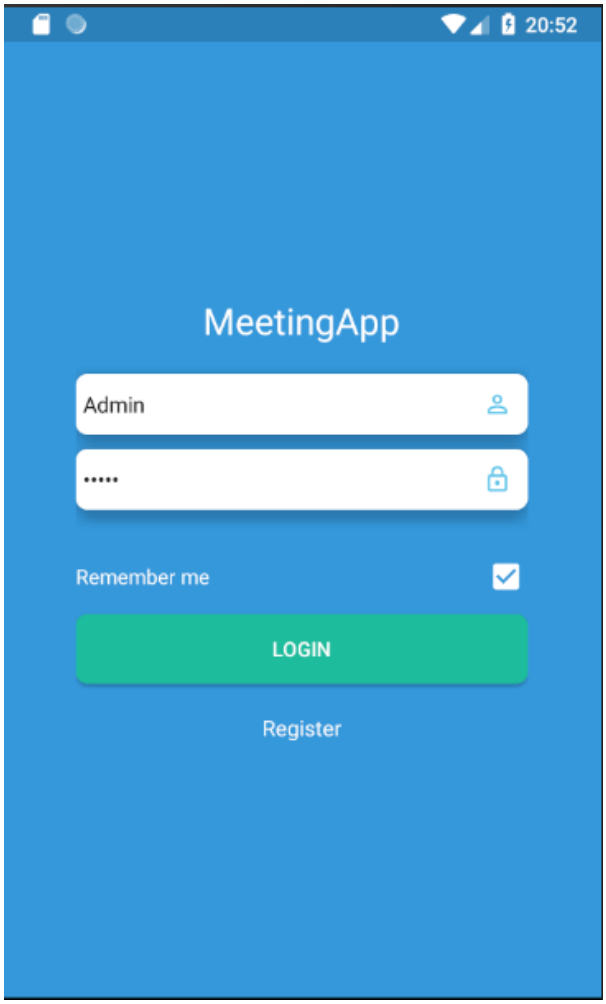
Työn tietokanta koostuu kuudesta taulusta. Näistä taulut meeting ja participant muodostavat moni moneen suhteen. Yhdessä kokouksessa voi olla osallisena useita osallistujia sekä yksi osallistuja voi olla osallisena useassa kokouksessa. Taulut meeting_participant_set ja participant_meeting_set muodostavat varsinaiset suhteet kokouksien ja osallistujien välille. Nämä suhteet luodaan käyttämällä edellä mainittuja primary-sekä foreign key-liitoksia.

”Applikation_user” taulussa on tallennettuna kaikki applikaation käyttäjät. Tässä taulussa salasanat on muutettu salattuun muotoon käyttämällä ns. hash -tekniikkaa. ”Meeting_file” -taulu sisältää puolestaan kaikki tietokantaan tallennetut tiedostot. Tallennetun tiedoston binäärikoodi on tallennettuna ”Binary Large Object” -tiedostona tauluun. Tiedostoilla on myös tieto user-kentässä siitä, kuka ko. tiedoston on luonut sekä tieto mille kokoukselle tiedosto on tallennettu, meeting_id-kentässä.

4.2 Mobiiliapplikaatio

Tässä osiossa kuvataan valmista mobiiliapplikaatiota. Siinä käydään läpi applikaation eri näkymät niiden loogisessa esiintymisjärjestyksessä.

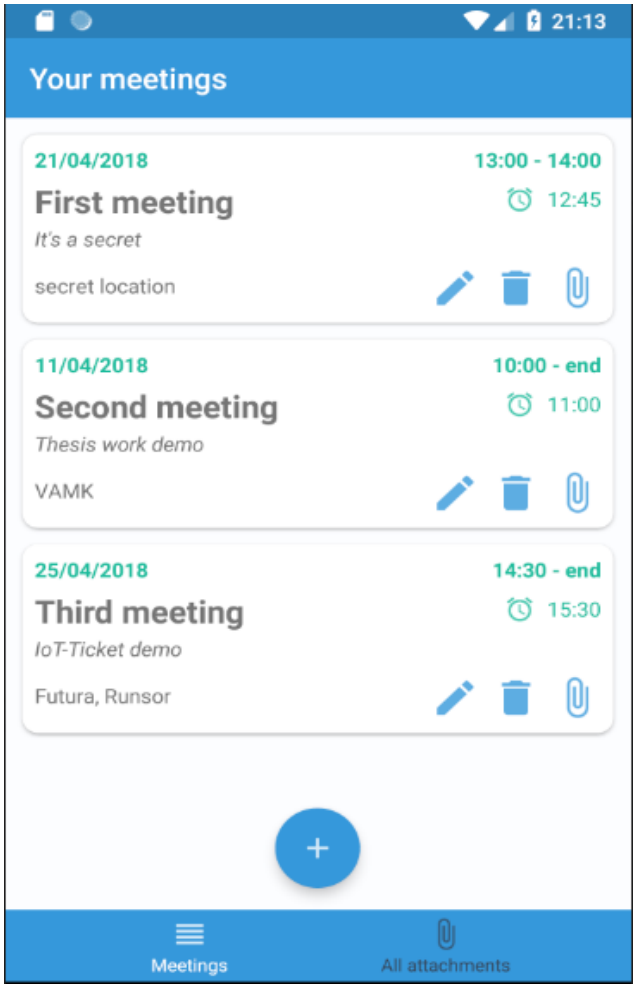
4.2.1 Login – sisäänkirjautuminen



Kuva 9. Sisäänkirjautuminen

Kun applikaatio kännistyy, näytetään käyttäjälle ensimmäisenä kuvan 9 näkymä. Tässä näkymässä käyttäjä kirjautuu sisään applikaatioon ja/tai rekisteröityy käyttäjäksi. Näkymässä on kentät käyttäjänimen sekä salasanan syöttämistä varten sekä sisäänkirjautumis -ja rekisteröintinapit. Rastittamalla valintaruudun, käyttäjätunnukset on mahdollista tallentaa seuraavan sisäänkirjautumisen nopeuttamiseksi. Tapauksessa, jossa käyttäjätunnus tai salasana on virheellinen, näytetään käyttäjälle virheilmoitus. Käyttäjä saa myös ilmoituksen, jos hän yrittää rekisteröityä varatulla käyttäjänimellä.

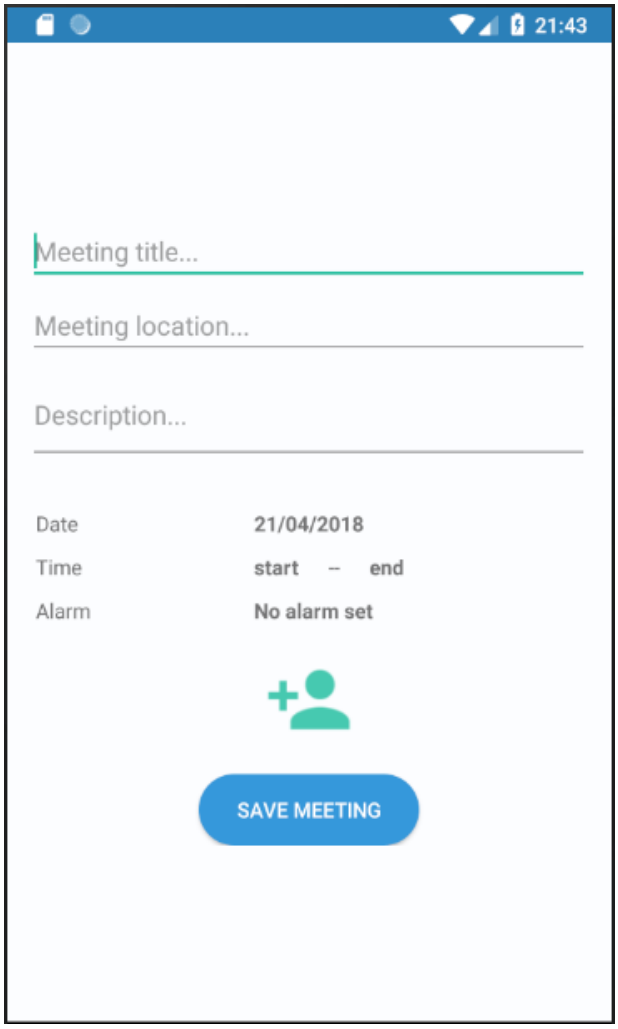
4.2.2 Your meetings – kokouslista



Kuva 10. Käyttäjäkohtaiset kokoukset

Kun applikaatioon on kirjauduttu sisään, avautuu käyttäjälle näkymä, jossa on hänen tallentamansa kokoukset. Tässä tapauksessa käyttäjä on tallentanut kolme kokousta, joissa jokaisessa on seuraavat tiedot: päivämäärä, aloitus -ja lopetusajankohta, otsikko, kuvaus, paikka sekä hälytyksen kellonaika. Jokaisella kokouksella on myös napit kokouksen muokkaamiseen, poistoon sekä kokouskohtaisten liitetiedostojen tallentamiseen palvelimelle. Plus-napilla aukeaa uusi näkymä uuden kokouksen luontiin ja tallennukseen.

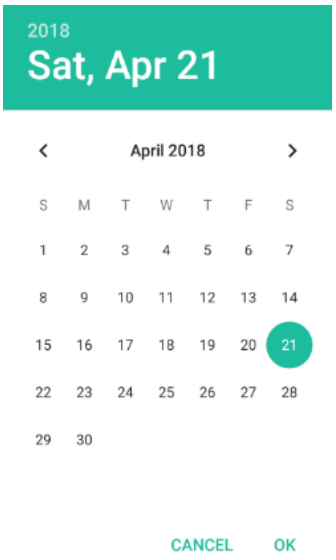
4.2.3 New meeting - kokouksen luoti ja tallennus



Kuva 11. Kokouksen luonti ja tallennus

Edellisen näkymän plus-nappia painamalla avautuu kuvan 11 mukainen näkymä uuden kokouksen luotiin sekä tallennukseen. Kokouksen otsikko-, paikka- sekä kuvaustiedot syötetään tekstikenttiin. Päivämäärälle, alku-ja lopetuskellonajalle, osallistujien lisäykselle sekä hälytykselle avautuvat ns. fragment -näkymät niitä klikatessa. Alla eriteltynä eri fragment -näkymät.

- Päivämäärän valinta



Kuva 12. Päivämäärän valinta fragment-näkymä

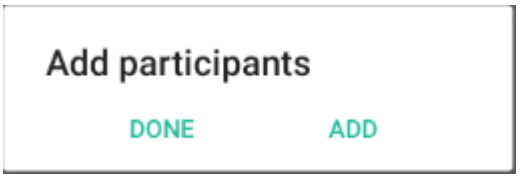
Päivämäärän valinnalle aukeaa yllä oleva näkymä. Käyttäjä valitsee päivämäärän ja painaa OK, jolloin valittu päivämäärä valitaan kokouksen päivämääräksi.

- Ajankohdan ja hälytyksen valinta

Kuva 13. Kellonajan valinta fragment-näkymä.

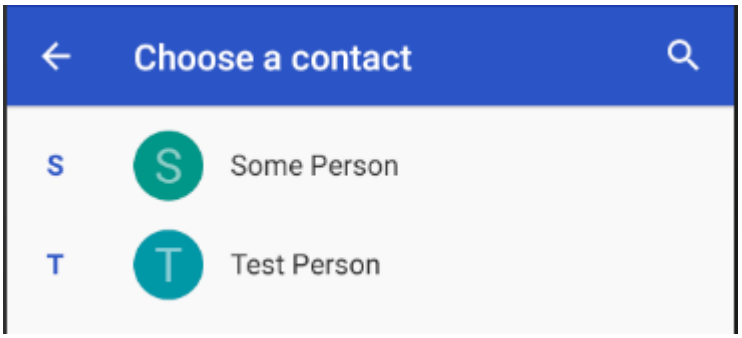
Kokouksen alkamis -ja loppumisajankohta valitaan kuvan 13 näkymällä. Myös hälytyksen valinta tehdään käyttämällä tätä samaa fragment-näkymää.

- Add participants - Osallistujien lisäys



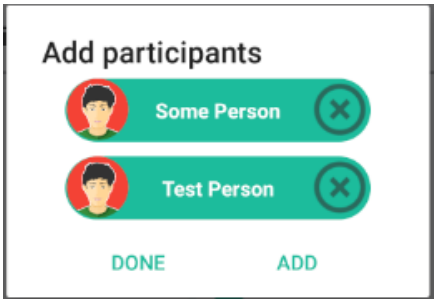
Kuva 14. Osallistujien lisäys

Kun käyttäjä klikkaa vihreää ikonia lisätäkseen osallistujan, aukeaa kuvan 14 näkymä. Koska lisättyjä osallistujia ei vielä ole, on näkymässä vain napit ”Done” ja ”Add”. ”Add” -nappia painaessa aukeaa puhelimen kontaktilista eli kaikki puhelimen omistajan kontaktit listattuna. (Kuva 15.)



Kuva 15. Puhelimen kontaktit

Tästä listasta käyttäjä voi valita osallistujia kokoukselleen. Kun käyttäjiä on lisätty, näyttää ”Add participants” -näkymä kuvan 16 mukaiselta.



Kuva 16. Kokoukselle lisätyt osallistujat

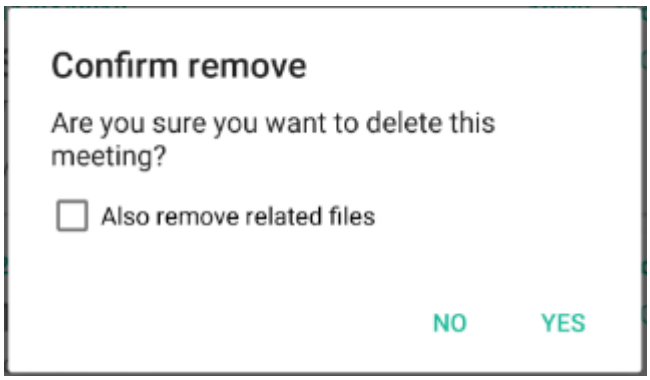
Kokoukselle on lisätty nyt kaksi osallistujaa. Käyttäjän voi poistaa painamalla käyttäjän nimen oikealla puolella olevaa rastia.

4.2.4 Edit meeting – kokouksen muokkaus

Jokaisella kokouksella on nappi ko. kokouksen muokkausta varten. Kokouksen muokkaus tapahtuu samassa ylläkuvatussa näkymässä kuin uuden kokouksen luonti. Erona on se, että ko. kokouksen tiedot on ladattu valmiiksi niille tarkoitettuihin kenttiin, jotta kokouksen muokkaus olisi helppoa ja nopeaa. Muokkauksen jälkeen kokous tallennetaan samaan tapaan kuin uuden luonti, eli ”Save meeting” -nappia painamalla.

4.2.5 Confirm remove – kokouksen poistodialogi

Kokous on mahdollista poistaa painamalla ko. kokouksen ”Roskakori” -nappia. Kun nappia painetaan, aukeaa kuva 17 mukainen poistodialogi.



Kuva 17. Kokouksen poistodialogi

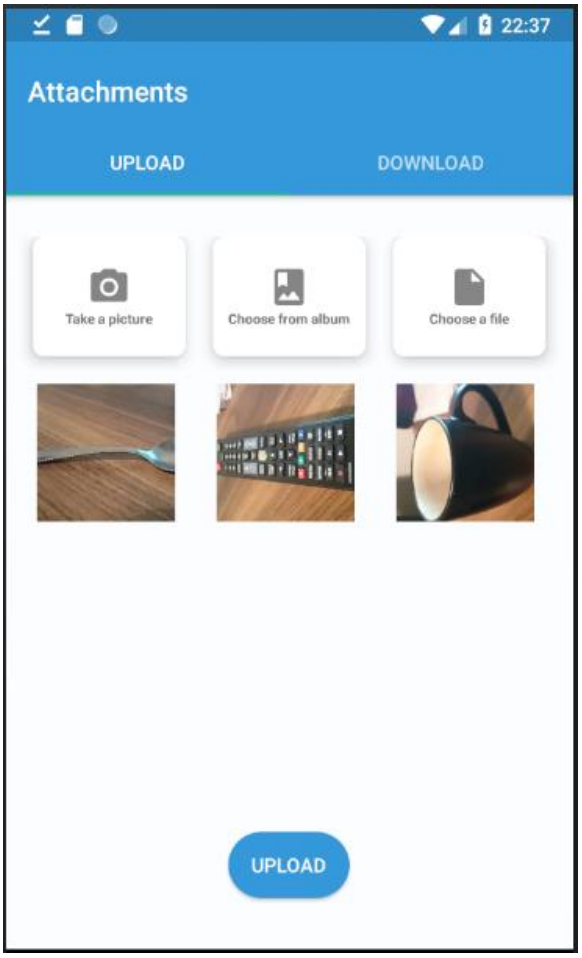
Käyttäjältä kysytään vahvistus kokouksen poistolle ennen kuin poisto suoritetaan. Samassa yhteydessä on myös mahdollista poistaa kokoukselle tallennetut liitetiedostot. Näin ne eivät jää roikkumaan palvelimelle tarpeettomasti.

4.2.6 Attachments – kokouskohtaisten liitetiedostojen tallennus ja lataus

Painamalla kokouksen ”Liitetiedosto” -nappia, aukeaa käyttäjälle näkymä kokouskohtaisten liitetiedostojen käsittelyyn. Tämä näkymä on jaettu kahteen

alinäkymään, Upload ja Download. ”Upload”-näkymässä tiedostoja ladataan palvelimeen ja ”Download”-näkymässä ladataan palvelimesta.

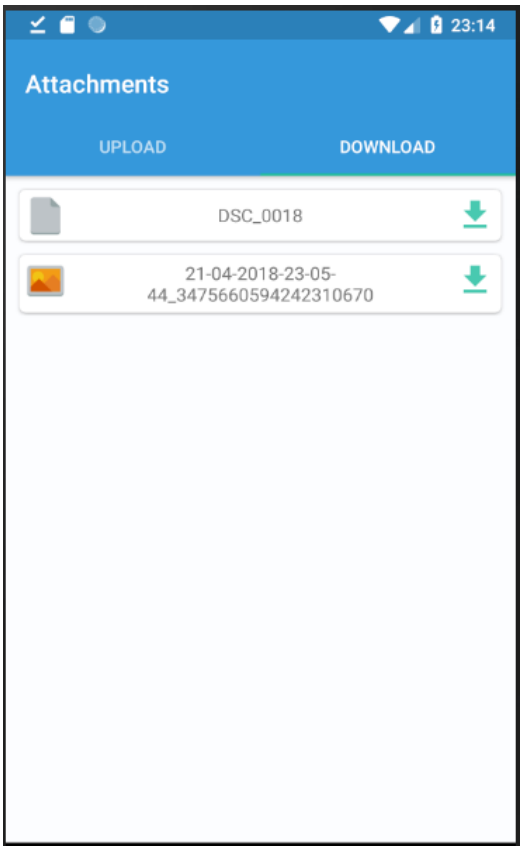
- Upload – palvelimeen tallennus



Kuva 18. Kokouskohtaisten liitetiedostojen tallennus

”Upload”-näkymässä liitetiedostoja on mahdollista valita kolmella tavalla. Käyttäjä voi ottaa kuvan puhelimen kameralla, valita jo otetun kuvan albumista tai valita tiedostoista jonkin halutun tiedoston. Valitut tiedostot näkyvät käyttäjälle ns. GridView -komponentissa. Kun käyttäjä on valinnut halutut tiedostot, käynnistetään tiedostojen tallennus palvelimeen painamalla ”Upload” -nappia.

- Download – palvelimesta lataus



Kuva 19. Kokouskohtaisten liitetiedostojen lataus

Tässä näkymässä tiedostoja voidaan ladata palvelimelta puhelimeen. Tiedoston lataus käynnistetään painamalla ko. tiedoston vihreää nuolta. Jokaisella tiedostolla on oma ikoni, riippuen tiedostomuodosta. Tämä auttaa käyttäjää löytämään nopeammin haluamansta tiedoston.

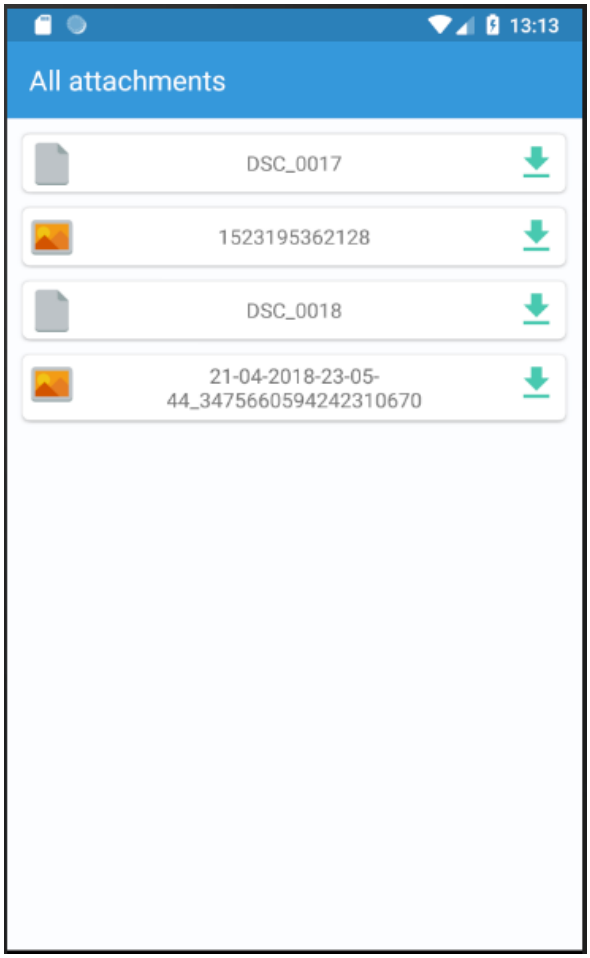
Tiedostoja on myös mahdollista poistaa palvelimelta. Tämä onnistuu pyyhkäisemällä haluttua tiedostoa vasemmalle, jolloin oikealle ilmestyy ”Roskakori” -ikoni poistoa varten. (Kuva 20.)

Kuva 20. Tiedoston poisto

4.2.7 All attachments – käyttäjäkohtaiset liitetiedostot

Kuva 21. Alavalikko

Applikaation-alavalikon ”All attachments”-valintaa painamalla, käyttäjä voi avata näkymän, jossa näkyy kaikki hänen palvelimelle tallentamansa liitetiedostot. Kuvassa 22 on kuvattu tätä näkymää.



Kuva 22. Käyttäjäkohtaiset liitetiedostot

Edellä kuvatussa ”Download”-näkymässä näkyi listattuna kaikki tiedostot, jotka oli tallennettu ko. kokoukselle. Tässä näkymässä on listattuna kaikki liitetiedostot, mitä nykyinen käyttäjä on tallentanut palvelimelle. Tämä näkymä helpottaa tietyn

liitetiedoston löytämistä, jos käyttäjä ei tarkalleen muista mihin kokoukseen haettava liitetiedosto kuuluu, varsinkin jos tallennettuja kokouksia on paljon.

Tässä näkymässä on jokaisella kokouksella samat toiminnot, mitä edellä mainitussa ”Download”-näkymässä; ko. tiedoston lataus sekä poisto.

5 YHTEENVETO

Mobiilisovelluskehityksestä minulla oli aikaisempaa kokemusta koulussa saamistani opinnoista yhden kurssin verran sekä omista projekteistani, mitä olen harrastuksena vapaa-ajallani tehnyt. Kuitenkaan en ollut koskaan ennen kehittänyt sovellusta, joka pystyy keskustelemaan palvelimen kanssa verkon yli, joten uusia asioita opin tätä työtä tehdessä runsaasti.

Tälle työlle asetetut vaatimukset saatiin kaikki täytettyä. Tulevaisuudessa aion kuitenkin jatkaa tämän projektin kehittämistä edelleen uusilla ominaisuuksilla. Yksi tällainen ominaisuus, mikä tuli työtä tehdessä mieleen, olisi offline-käyttömahdollisuus. Käyttäjä voisi luoda ja tallentaa tehtyjä kokouksia väliaikaisesti puhelimensa muistiin ja synkronoida offline-tilassa luodut kokoukset myöhemmin palvelimelle. Tällä hetkellä käyttäjän on oltava yhteydessä palvelimeen käyttääkseen applikaatiota.

Olen myös harkinnut tämän työn uudelleenkirjoittamista React Native-kirjastolla. React Native on mobiilikehitykseen tarkoitettu JavaScript -kirjasto. Hyödyntämällä tätä kirjastoa saisin kehitettyä tämän applikaation samalla lähdekoodilla myös iOS-käyttöjärjestelmää käyttäville iPhone-puhelimille. Jos aion jossain vaiheessa julkaista tämän sovelluksen yleiseen käyttöön, olisi tällöin hyvä, jos applikaatio olisi kehitetty kummallekin alustalle, jotta käyttäjäkunta olisi mahdollisimman suuri.

Minulla ei ollut aikaisempaa kokemusta REST -rajapintojen luomisesta, joten tämä olikin hyvä tilaisuus luoda ensimmäinen RESTful web-palveluni. Nykyajan web-sovelluskehityksessä REST -arkkitehtuuri on hyvin olennainen osa insinöörin perustaitoja, joten tätä työtä tehdessä oppimani taidot tulevat tarpeeseen jokapäiväisessä työelämässä.

Tällä hetkellä tämän työn palvelinohjelmisto pyörii paikallisesti omalla koneellani, jolla olen kehittänyt ohjelmistoa. Mahdollistaakseni mobiiliapplikaation julkaisun yleiseen käyttöön, on palveliohjelmisto laitettava pyörimään jossain julkisessa pilvipalvelussa. Nämä pilvipalvelut ovat osa

nykyaikaista mobiilikehitystä, jotka mahdollistavat suuret käyttäjäkunnat applikaation käyttöön. Näillä pilvipalveluilla on myös erilaisia työkaluja tietokantojen ylläpitoon sekä käyttäjäkohtaisten tiedostojen tallennukseen. Yksi mahdollinen pilvipalvelu johon tämän julkaisun voisin tehdä, on Microsoft Azure. Tällaiset palvelut ovat kuitenkin maksullisia, joten tätä opinnäytetyötä varten tyydyin kehittämään ohjelmistoa vain paikallisesti.

LÄHTEET

/1/ REST. Viitattu 9.4.2018. https://fi.wikipedia.org/wiki/REST

/2/ Representational state transfer. Viitattu 9.4.2018. https://en.wikipedia.org/wiki/Representational_state_transfer

/3/ Spring Security Reference, 5.8 Authentication. Viitattu 10.4.2018. https://docs.spring.io/spring-security/site/docs/5.0.3.RELEASE/reference/htmlsingle/#jc-authentication

/4/ Spring Docs. Data Access. Viitattu 11.4.2018 https://docs.spring.io/spring/docs/current/spring-framework-reference/data-access.html

/5/ Spring Docs. 5. The IoC container. Viitattu 12.4.2018 https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html

/6/ Spring Docs. 17. Web MVC framework. Viitattu 12.4.2018 https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html

/7/ Spring Docs. 43. Testing. Viitattu 15.4.2018. https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-testing.html

/8/ Spring Boot homepage. Viitattu 16.4.2018 https://projects.spring.io/spring-boot/

/9/ Spring Docs. 9.1 Servlet Containers. Viitattu 17.4.2018 https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#getting-started-system-requirements-servlet-containers

/10/ Retrofit homepage. Viitattu 18.4.2018 http://square.github.io/retrofit/